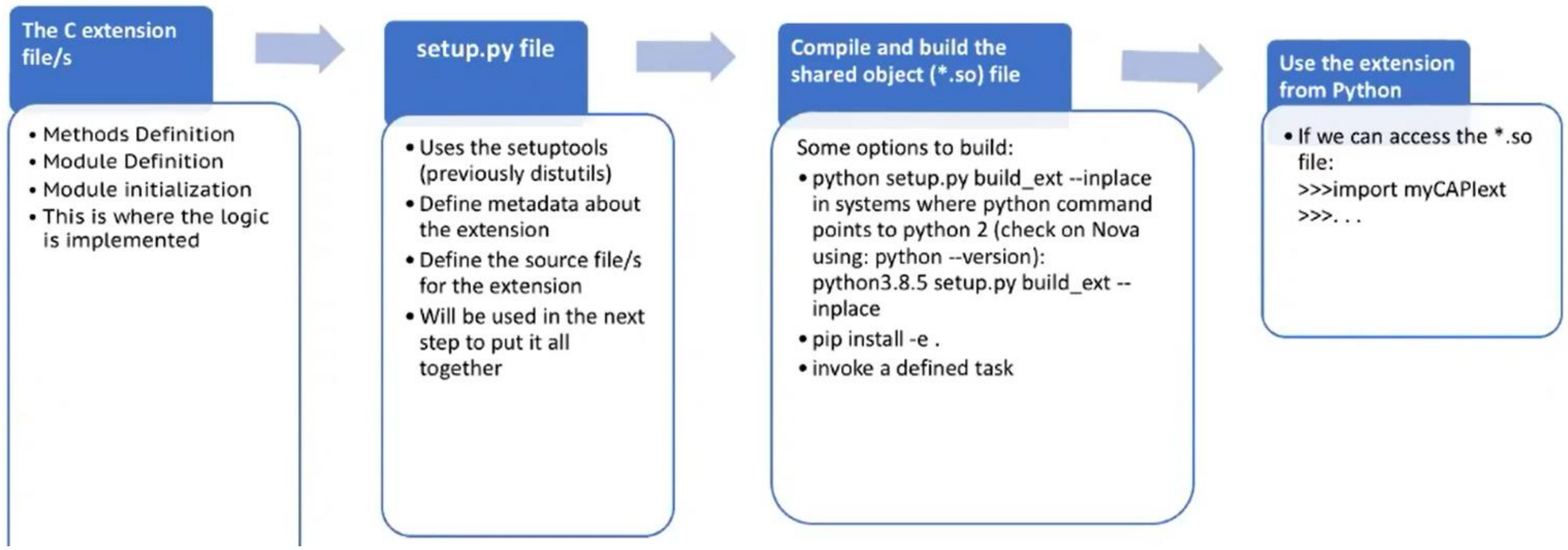
The Python C-API

- This is an extension module that requires several dozen lines of C code, most of it boilerplate that calls the Python/C API
- When calling from Python, the extension module must convert Python objects to C data, compute (and if we want convert the result back to a Python object)
- Very good documentation - https://docs.python.org/3.8/c-api/index.html



The C extension file/s

- Methods Definition
- Module Definition
- Module initialization
- This is where the logic is implemented

setup.py file

- Uses the setuptools (previously distutils)
- Define metadata about the extension
- Define the source file/s for the extension
- Will be used in the next step to put it all together

Compile and build the shared object (*.so) file

Some options to build:

- python setup.py build_ext --inplace in systems where python command points to python 2 (check on Nova using: python --version): python3.8.5 setup.py build_ext --inplace
- pip install -e .
- invoke a defined task

Use the extension from Python

- If we can access the *.so file:

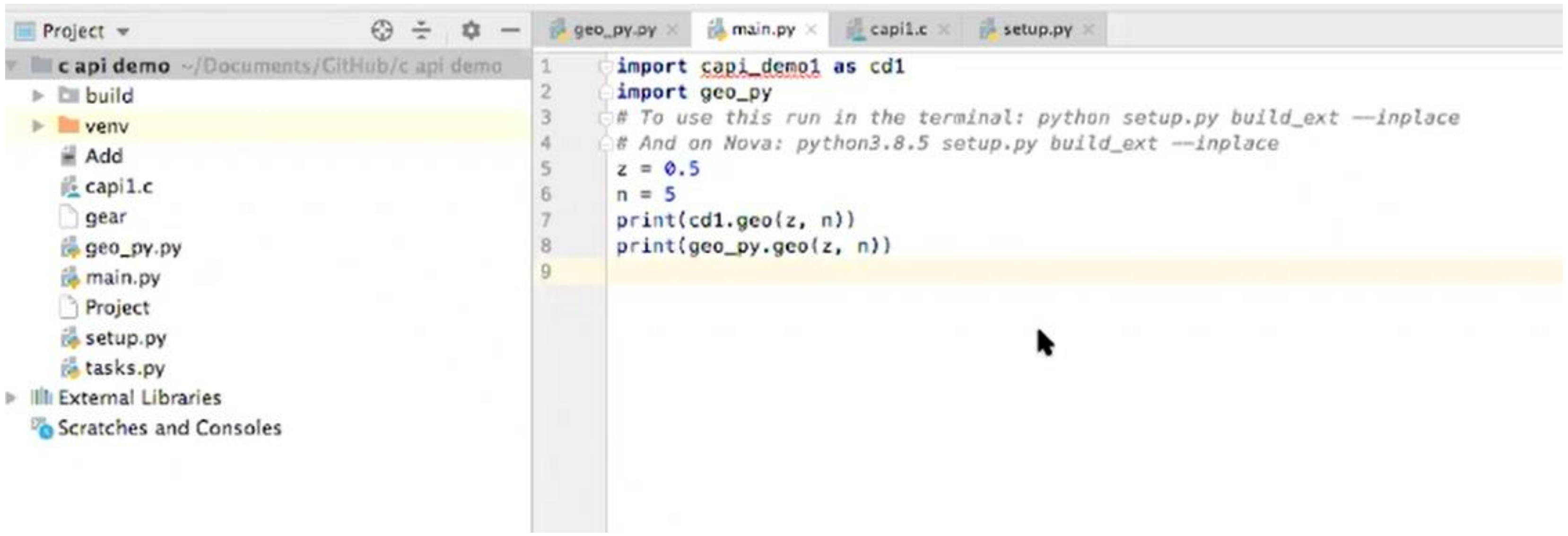
```
>>>import myCAPIext
>>>. . .
```

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots = \sum_{n=0}^{\infty} \frac{1}{2^n} = 2.$$

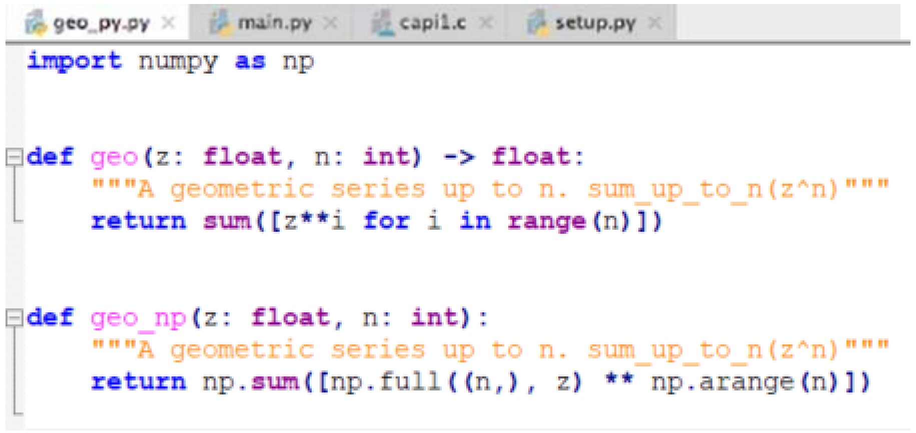
In general, the geometric series

$$\sum_{n=0}^{\infty} z^n$$

converges if and only if $|z| < 1$.



```python
import capi_demo1 as cd1
import geo_py
# To use this run in the terminal: python setup.py build_ext --inplace
# And on Nova: python3.8.5 setup.py build_ext --inplace
z = 0.5
n = 5
print(cd1.geo(z, n))
print(geo_py.geo(z, n))
```

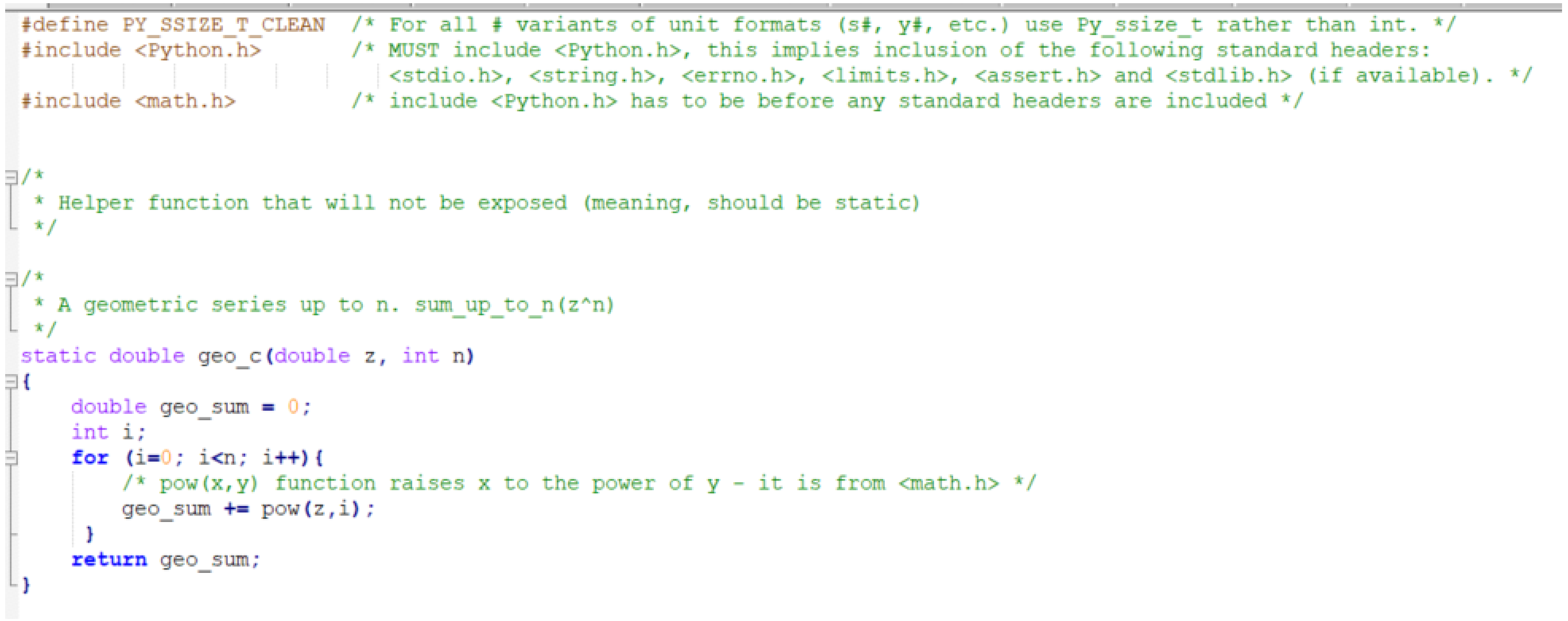


```python
import numpy as np


def geo(z: float, n: int) -> float:
    """A geometric series up to n. sum_up_to_n(z^n)"""
    return sum([z**i for i in range(n)])


def geo_np(z: float, n: int):
    """A geometric series up to n. sum_up_to_n(z^n)"""
    return np.sum([np.full((n,), z) ** np.arange(n)])
```

capi1.c

Always first two lines

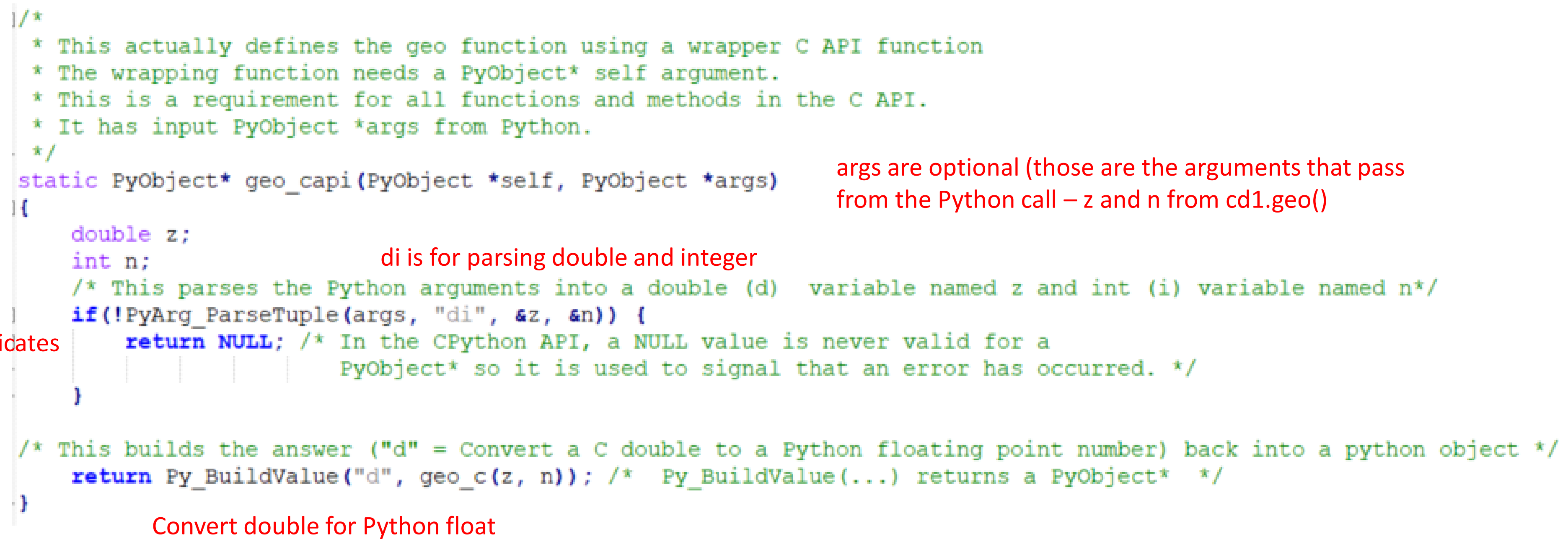
Must be static
A simple C function that makes the computation



```c
#define PY_SSIZE_T_CLEAN  /* For all # variants of unit formats (s#, y#, etc.) use Py_ssize_t rather than int. */
#include <Python.h>       /* MUST include <Python.h>, this implies inclusion of the following standard headers:
                             <stdio.h>, <string.h>, <errno.h>, <limits.h>, <assert.h> and <stdlib.h> (if available). */
#include <math.h>         /* include <Python.h> has to be before any standard headers are included */


/*
 * Helper function that will not be exposed (meaning, should be static)
 */

/*
 * A geometric series up to n. sum_up_to_n(z^n)
 */
static double geo_c(double z, int n)
{
    double geo_sum = 0;
    int i;
    for (i=0; i<n; i++){
        /* pow(x,y) function raises x to the power of y - it is from <math.h> */
        geo_sum += pow(z,i);
    }
    return geo_sum;
}
```

This is a regular C function to compute what we need

capi1.c



```c
/*
 * This actually defines the geo function using a wrapper C API function
 * The wrapping function needs a PyObject* self argument.
 * This is a requirement for all functions and methods in the C API.
 * It has input PyObject *args from Python.
 */
static PyObject* geo_capi(PyObject *self, PyObject *args)
{
    double z;
    int n;
    /* This parses the Python arguments into a double (d)  variable named z and int (i) variable named n*/
    if(!PyArg_ParseTuple(args, "di", &z, &n)) {
        return NULL; /* In the CPython API, a NULL value is never valid for a
                        PyObject* so it is used to signal that an error has occurred. */
    }

/* This builds the answer ("d" = Convert a C double to a Python floating point number) back into a python object */
    return Py_BuildValue("d", geo_c(z, n)); /*  Py_BuildValue(...) returns a PyObject*  */
}
```

args are optional (those are the arguments that pass from the Python call – z and n from cd1.geo()

di is for parsing double and integer

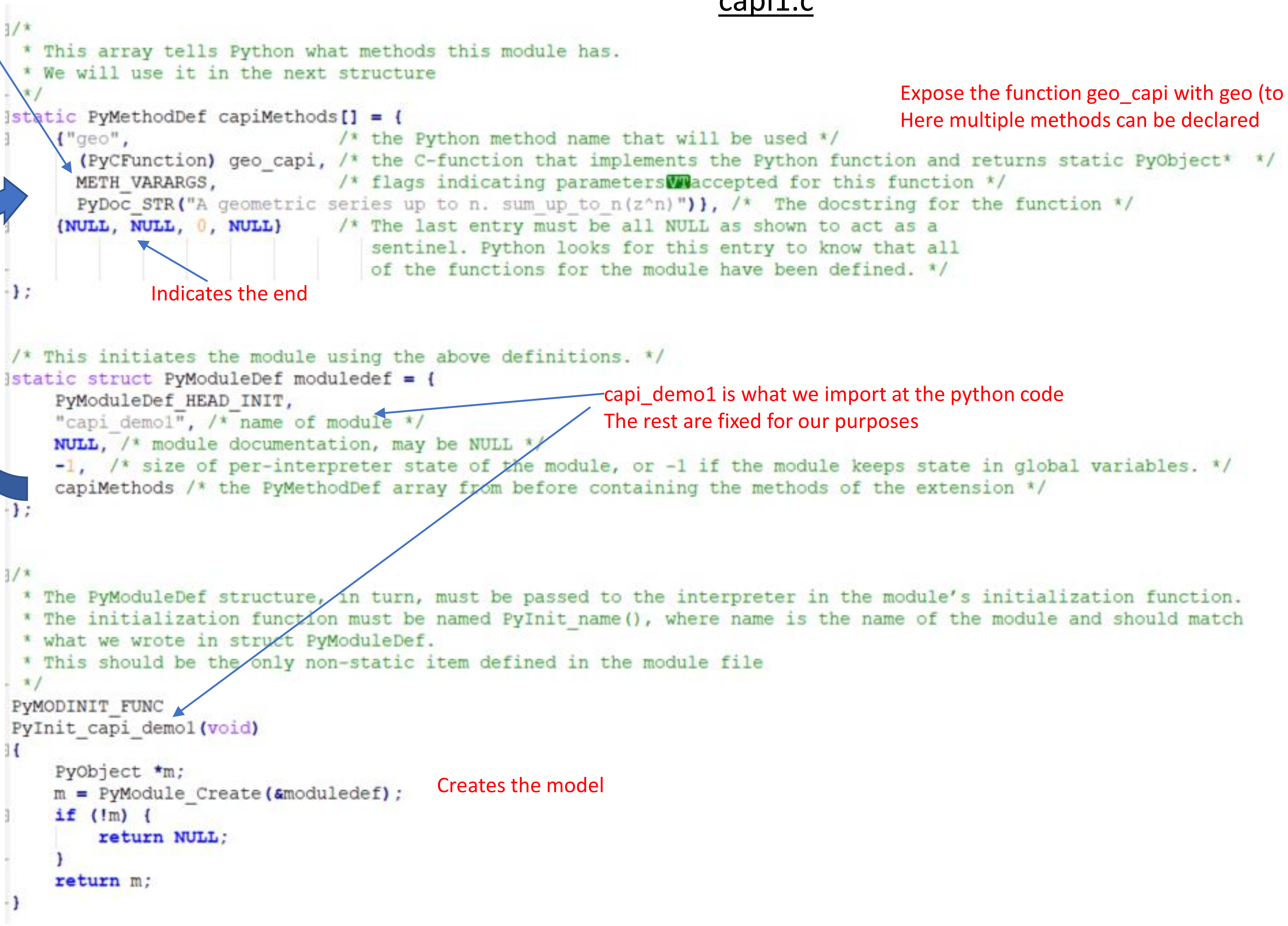
NULL indicates error

Convert double for Python float

This is the main function to connect C and Python

capi1.c

States that the functions gets arguments



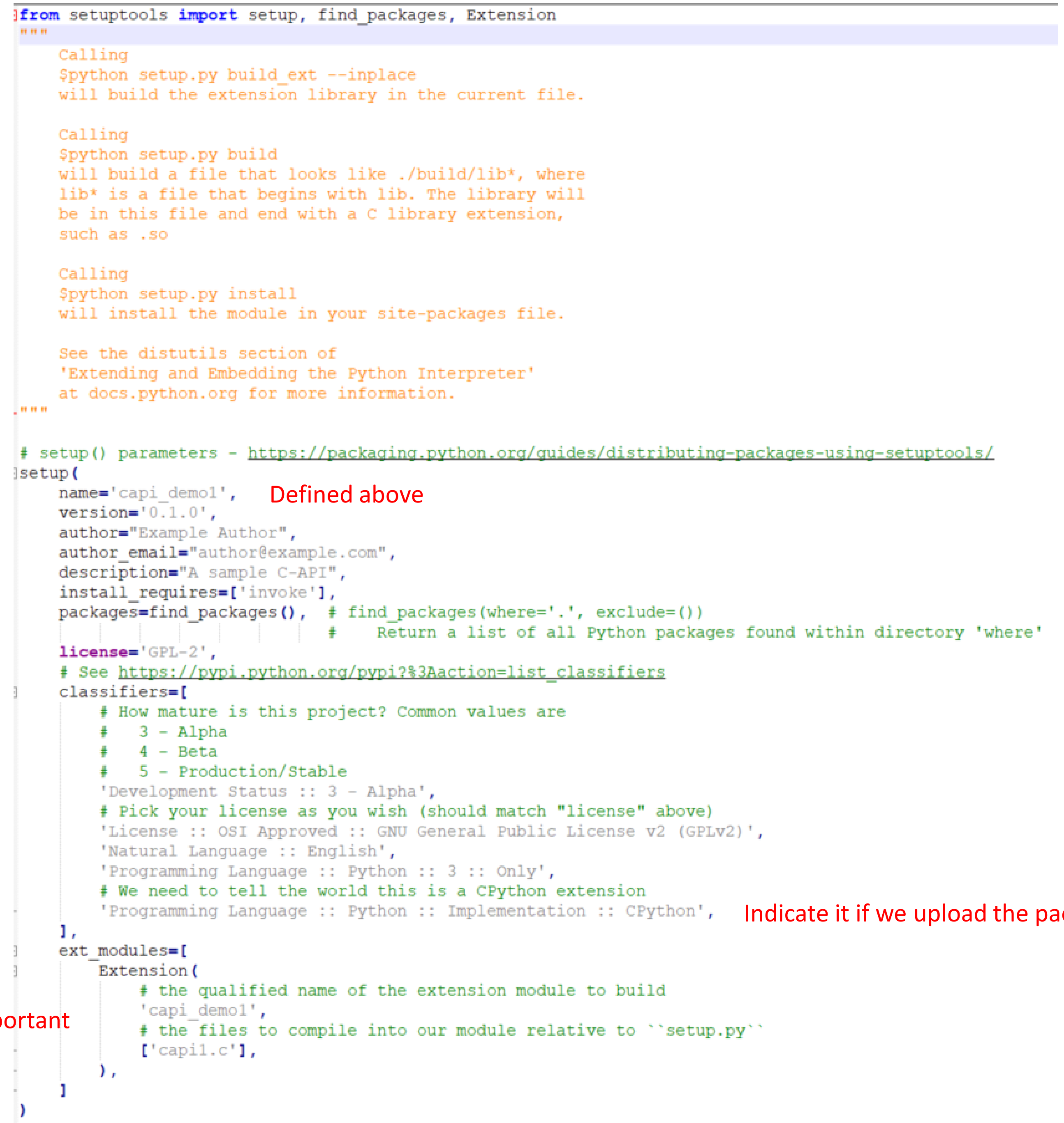
```c
/*
 * This array tells Python what methods this module has.
 * We will use it in the next structure
 */
static PyMethodDef capiMethods[] = {
    {"geo",                  /* the Python method name that will be used */
     (PyCFunction) geo_capi, /* the C-function that implements the Python function and returns static PyObject*  */
     METH_VARARGS,           /* flags indicating parametersVTaccepted for this function */
     PyDoc_STR("A geometric series up to n. sum_up_to_n(z^n)")}, /*  The docstring for the function */
    {NULL, NULL, 0, NULL}    /* The last entry must be all NULL as shown to act as a
                                sentinel. Python looks for this entry to know that all
                                of the functions for the module have been defined. */
};


/* This initiates the module using the above definitions. */
static struct PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT,
    "capi_demo1", /* name of module */
    NULL, /* module documentation, may be NULL */
    -1,  /* size of per-interpreter state of the module, or -1 if the module keeps state in global variables. */
    capiMethods /* the PyMethodDef array from before containing the methods of the extension */
};


/*
 * The PyModuleDef structure, in turn, must be passed to the interpreter in the module's initialization function.
 * The initialization function must be named PyInit_name(), where name is the name of the module and should match
 * what we wrote in struct PyModuleDef.
 * This should be the only non-static item defined in the module file
 */
PyMODINIT_FUNC
PyInit_capi_demo1(void)
{
    PyObject *m;
    m = PyModule_Create(&moduledef);
    if (!m) {
        return NULL;
    }
    return m;
}
```

Expose the function geo_capi with geo (to be called from python with cd1.geo()
Here multiple methods can be declared

Indicates the end

capi_demo1 is what we import at the python code
The rest are fixed for our purposes

Creates the model



```python
from setuptools import setup, find_packages, Extension
"""
    Calling
    $python setup.py build_ext --inplace
    will build the extension library in the current file.

    Calling
    $python setup.py build
    will build a file that looks like ./build/lib*, where
    lib* is a file that begins with lib. The library will
    be in this file and end with a C library extension,
    such as .so

    Calling
    $python setup.py install
    will install the module in your site-packages file.

    See the distutils section of
    'Extending and Embedding the Python Interpreter'
    at docs.python.org for more information.
"""

# setup() parameters - https://packaging.python.org/guides/distributing-packages-using-setuptools/
setup(
    name='capi_demo1',
```

Defined above

```python
    version='0.1.0',
    author="Example Author",
    author_email="author@example.com",
    description="A sample C-API",
    install_requires=['invoke'],
    packages=find_packages(),  # find_packages(where='.', exclude=())
                               #    Return a list of all Python packages found within directory 'where'
    license='GPL-2',
    # See https://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[
        # How mature is this project? Common values are
        #   3 - Alpha
        #   4 - Beta
        #   5 - Production/Stable
        'Development Status :: 3 - Alpha',
        # Pick your license as you wish (should match "license" above)
        'License :: OSI Approved :: GNU General Public License v2 (GPLv2)',
        'Natural Language :: English',
        'Programming Language :: Python :: 3 :: Only',
        # We need to tell the world this is a CPython extension
        'Programming Language :: Python :: Implementation :: CPython',
```

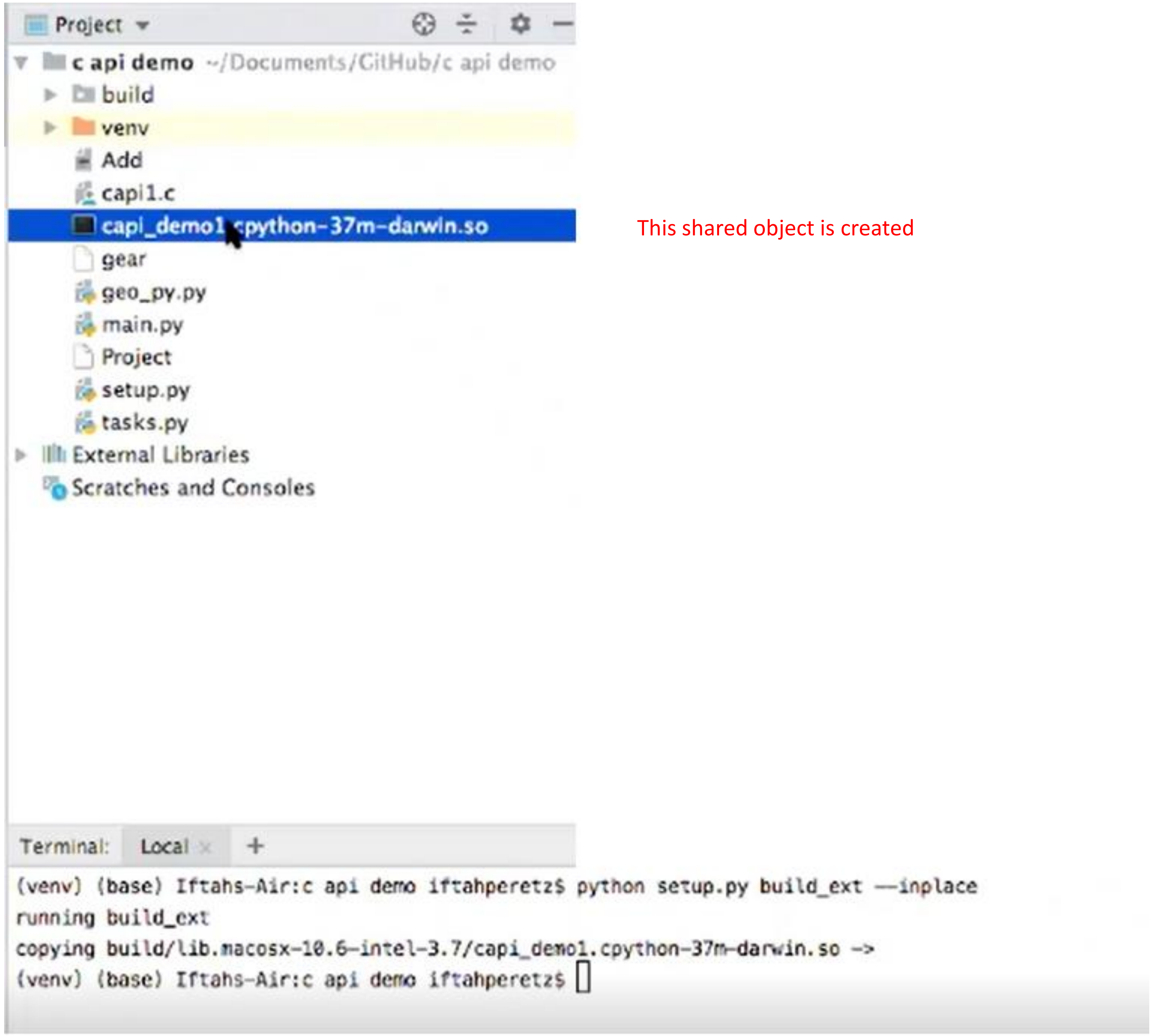
Indicate it if we upload the package online

Important

```python
    ],
    ext_modules=[
        Extension(
            # the qualified name of the extension module to build
            'capi_demo1',
            # the files to compile into our module relative to ``setup.py``
            ['capi1.c'],
        ),
    ]
)
```

Next we execute:
python setup.py build_ext –inplace

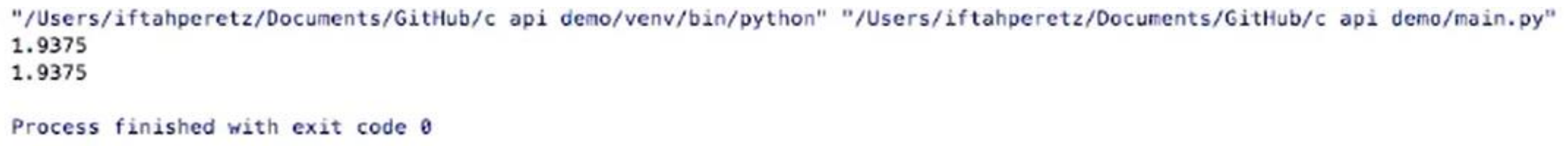
And then simply run the Python code



```
(venv) (base) Iftahs-Air:c api demo iftahperetz$ python setup.py build_ext —inplace
running build_ext
copying build/lib.macosx-10.6-intel-3.7/capi_demo1.cpython-37m-darwin.so ->
(venv) (base) Iftahs-Air:c api demo iftahperetz$
```

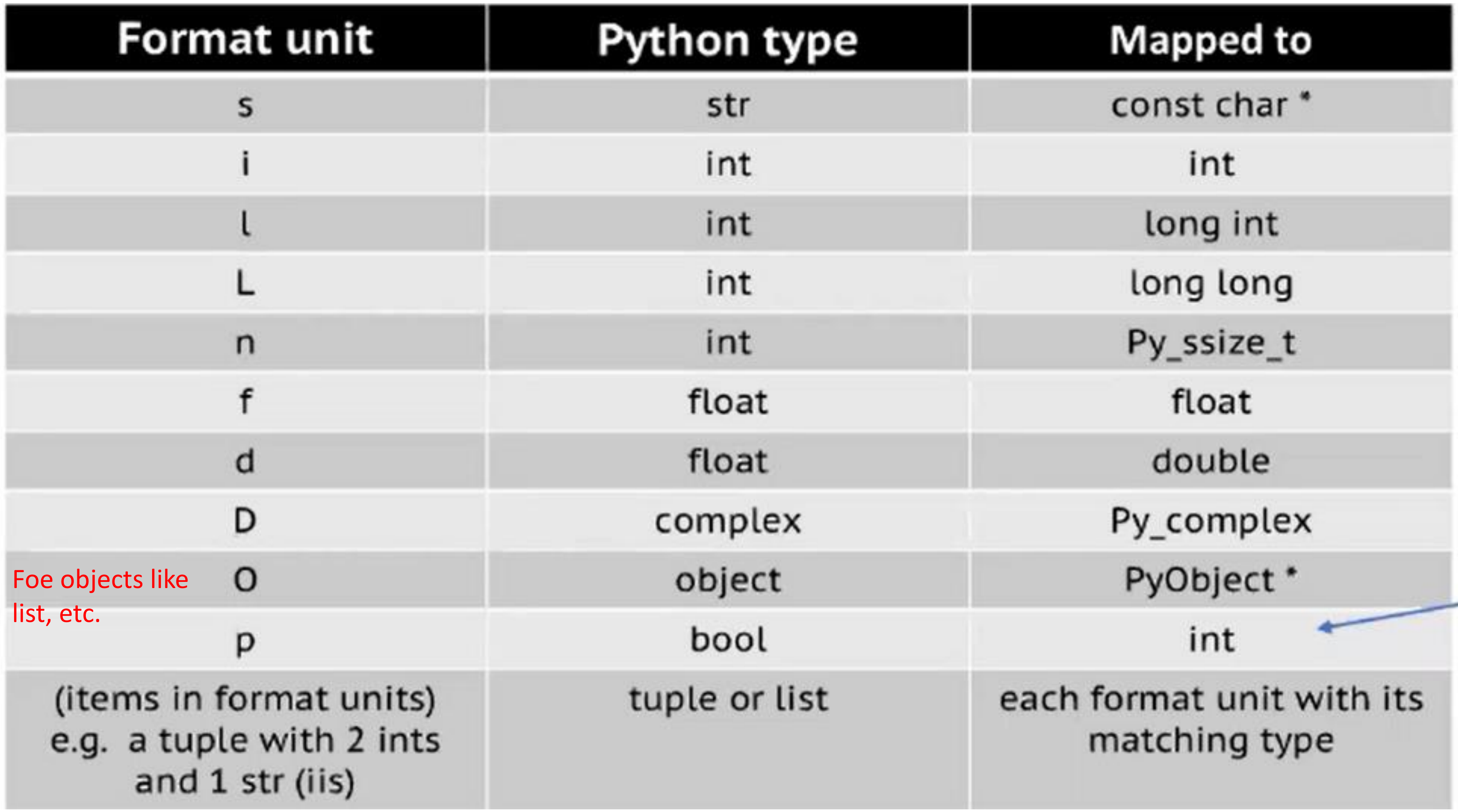
This shared object is created

And then simply run the Python code



```
"/Users/iftahperetz/Documents/GitHub/c api demo/venv/bin/python" "/Users/iftahperetz/Documents/GitHub/c api demo/main.py"
1.9375
1.9375

Process finished with exit code 0
```

The datatype conversions



Format unit	Python type	Mapped to
s	str	const char *
i	int	int
l	int	long int
L	int	long long
n	int	Py_ssize_t
f	float	float
d	float	double
D	complex	Py_complex
O	object	PyObject *
p	bool	int
(items in format units) e.g. a tuple with 2 ints and 1 str (iis)	tuple or list	each format unit with its matching type

Foe objects like list, etc.

1 if the expression was True and 0 if it was False